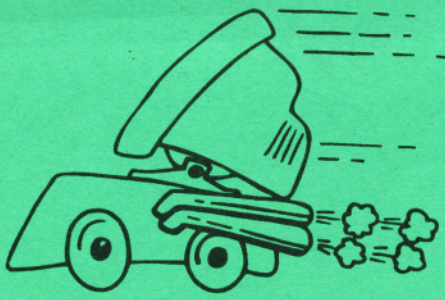
SAM Vision for SAM C



FRED PUBLISHING

SAM VISION

The SAM vision library is a very powerful library for use with SAM C. The library (SV) is an I8K block of machine code that implements many procedures, for example; printer support, dialogue boxes, mouse controls, disc file support, 16 and 19 bit arithmetic etc... SV is 100% machine code and works very fast. Using SV it is possible to develop your own applications for SAM. SV does not use the SAM ROM, it uses hookcode from MDOS. All your programs that you develop using SV can be freely spread with the SV40.LIB file. Providing you clearly mention SAM C & SAM VISION have been used.

SYSTEM DETAILS

SV is a fully relocatable code block with steps of 16384 bytes (16 K). SV must be started from a page in range 1 (implicit in SAM C) to 27 (&8000 to &70000 in hex). When SV runs, the code is paged into sections A & B of memory (at address from &0000 to &4a00). The spare room from &4800 to &7fff is used for the stack pointer. Space from &4a00 &7fff(in (in second SV page) and from &8000 to &7fff in other pages is free for your code and data. You can modify HMPR arbitrary. Data in sections C and D uses SV 19 bit address representation.

Unlike SAM C, SAM VISION can handle numbers greater than 65535. To store higher numbers, the 19bit number representations are used. On SAM, 19bit numbers (and addresses) are stored in the form of a structure.

SAM Vision was programmed by Marlin Krivos.
Testing by Steven Ekins.
Manual by Marlin Krivos and Steven Ekins.

19 Bit representation,

There are 2 forms of 19 bit representation, one is used for addresses, the other for numbers.
19 bit addresses use 3 bytes of memory, and are in the form of.

```
typedef struct
char apage;        //range 0..31
unsigned aoffset; // range 32768 .. 49151
} FAR;
```

The 19 bit number representation is similar to this.
To distinguish between 19 size and address, the data type BIG is used for 19 bit numbers.
19bit numbers can be set using SV functions, you can also print them and perform arithmetic operations on them.
As well as the BIG and FAR data types, these other types are also used.

```
typedef char * STRING; //strings
typedef int BOOL;
typedef unsigned UINT;
typedef UINT WORD;
typedef unsigned char BYTE;
typedef BYTEUCHAR;
typedef FILENAME; (see file functions).
```

The FAR structure is used by many SV functions (that use all of RAM). Here is a simple map of memory:
00004a00 800ffff
- is SV code,
- free memory
- free memory and free paging part of RAM.

Running a Sam Vision program,

Applications that run under SV are executed from the address where the SV40.LIB file is loaded, (at start of page e.g. 32768, 49152, 65536 ...). SV is initialised then the application is executed. The application is loaded at address start_of SV+&4a00 and is assembled at address &4a00. When your application is terminated, SV closes dialogue boxes and memory. SV is an interface between SAM and your applications. You can turn SV on and off from the option/preferences menu. When SV is being used, at the start of your program you should have : #include "stdsv.h" in place of "stdio.h". "runtime2.s" is used in place of "runtime.s". These files contain all of the SV prototypes etc...

How to load your own application ...
1. You should study some examples on disc first.

2. Use the SVloader tile to load the code..
a) change the string variable name$ to file name of your object file
b) set variable spare to 2 if object file (plus other files loaded behind the object file) has length <= 13824 byte. You add one for every extra 16384 bytes of length. E.g spare=3 if 13824 <= length <= 30208 etc.

d) save this basic under new name with autostart from line 30.

Other information about SAM Vision interface ..

The SV works with 256KB RAM & TAPE. It also works with drive 1, 2 or (if MDOS loaded) RAMDISC, it uses CENTRONICS port at addresses #E8 & #E9, Does not use SAM BUS, SAM CLOCK. The interface has been written in machine code and is compatible with MASTER BASIC.

MEMORY

The SV40.LIB file can be loaded in every page besides no. 0,2, DOS PAGE and VIDEO ('AGES (implicit is page no.1 - address 32768) and is allocated two pages (start page and next page). The M/C is assembled from address &0000 to address &49FF) and for your own code there is space from &4A00 to &7FFF. In the spare from &8000 to &FFFF you can situate any RAM page. When SV is initialized, it looks at the system table ALLOCT and allocates used pages. Then it searches for first and last free page in your SAM (by alloct). If in your SAM there are less than three pages free, then system won't be executed !!! This is an ideal memory map:

```
                ( AT 256 KB machine )
                  basic      free          SV dos screen TABLE
40 40 40 40 00 00 00 00 00 00 00 53 56 CO 60 60 NUMBER 00
01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
```

In this case the system uses RAM from page 4 to page 10 like a buffer for screen save and heap for memory sharing. The buffer moves to high addresses, and heap moves to low addresses. Spare between buffer and heap is accessible by TestSpace()

VIDEORAM

No SV funcs change VIDEO page (VMPR). PRINT routine works in mode 3 and 4. Palette is updated every 0.02 sec.

INTERRUPT

Interrupts are handled from FRAME and LINE only. When FRAME, palette is updated, call music module if it's necessary. LINE vector is NULL and you can redefine it. New int. handler is available:

```
      LD HL,(#3A); redirect of interrupt
      LD  (END+I),HL  LD
      HL,NEWINT       LD
      (#3A),HL
      RET
NEWINT: ...
```

```
    ...         ; new handle
END: J OLDINT           ; terminate, jump to old handler
```

KEYBOARD

The differences from standard ROM routine:

1. doesn't use buffer for key pressing

2. ESC pressing returns value 255

PRINTER

You can redefine standard init string (15 byte long) to initialize it.
Output is in two modes - text and graphics (with standard BASIC font)

SOUND

Modules from Etracker or ProTracker are available. NB : Although provision has been made for ProTracker, it has not been completed as yet. ProTracker is hoped to be released later in 1996.

FILES

You can load directory of disc to buffer and use this information. SV uses directories up to 80 items. Every file item is 32 bytes long:

Offset Mark
00 file type (bit 0 .. 4)
bit 6 is PROTECT ON/OFF
bit 5 is HIDE ON/OFF
01..10 file name
11 flag - unused
12 start track of file
13 start sector of file
14..16 numbers of used sectors
17..19 PAGE/OFFSET address of start file in RAM
20..22 PAGE/OFFSET form of file size
23..25 address of execution
etc.

COLOUR

The system has only one colour variable. You can change it by Colour(). Windows drawing change current colour to your own, colour. When mode 3 is set, you can use colour no. 0,5,10 and 15 only.

STACK

When the system is started, the stack pointer is set to address &4A00. Spare for it is cca 1024 byte.

About the menu controls...

When the arrow pointer is displayed, You can press ESC to cancel, also you can click outside the current window to cancel. When the input is displayed, pressing ESC will cancel the operation without changes, pressing EDIT will clear the line, RETURN terminates operation, DEL & ARROWS KEY as per BASIC ...

SAM VISION FUNCTIONS.

File handling functions

Sam vision allows you to work with drive 1, 2 or 3 (RAM disc). Although the file functions will work with SAMDOS, MasterDOS is strongly recommended.

Many of these functions will require the use of the **FILENAME** structure. The format of the structure is:

```
typedef struct {
        unsigned char fntype;       // file type
        char fnname[10];
         } FILENAME;
```

fntype is the type of file (16 =Basic,... see tech manual.).
fnname is the filename 10 chars, the wildcards (*and ?) are available.

int Load (FILENAME *file);
This function loads a file at the address of the header. Returns 0 if no errors occurred, else returns the error number.

int LoadAt (FILENAME *file, FAR *a);
The same as load but you can specify the load address.
Returns 0 if no errors occurred, else returns the error number.

An example file loading program.

```
#include "stdsv .h"


main()
  {
FILENAME f; // filename structure
FAR a;              // address structure
f.fntype=CODE; //set file type to code
Move ("myfile ", f.fnname, 10);
a.apage=3;
a.aoffset=32768; //set page and offset
LoadAt (&f, &a);}
```

This example loads a code file called "myfile" at address 65536 (start of page 3).

int Save (FILENAME *n, far *start, BIG *length);
This is the same as the basic save command. it writes a block of memory from address start, with length, length.

int Restore (void);
Sets the drive head to track 0, returns the result.

void Device (int drive);
Sets the drive to be used, 1, 2 or 3(ramdisk).

int Erase (FILENAME *file);
Erases the file with name file.fnname. The type of file is irrelevant, the wildcards (* and ?) are not available.

unsigned Write (int track, int sector, unsigned addr);
unsigned Read (int track, int sector, unsigned addr);
These functions are used to read and write data directly to the current disk. You can use address (&4a00...&ffff). SV uses its own handlers for the drive. The result is addr+512 if ok, else NULL if an error occurred.

DSTAT *Dstat (int drive);
This function is important for reading disc directories. Dsdat () reads and saves into a FileTable buffer information about the first 80 files from the disc. FileTable is an array of 80 file structures. The result is a pointer to another structure DSTAT, that describes the current disc. Null is returned if an error occured when reading the directory. Parameter is the drive number. Use 0 for reading the current drive. SV checks access to the disc, if the data is already in memory then the function returns without reading. This is used to speed up the function, add 128 to the param to switch it off.

if drive =
0 - load from current disk
3 - load from disk no 2
128 - load from current disk without checking
128+1 - load from disk 1 without checking etc..

Description of DSDAT and FILE structures.

```
typedef struct{
        char dname[10]; //disk name
          unsigned char dtracks; // number of tracks of directory (4)
          int dident; //identification code of disk
          } DSTAT

typedef struct {
        char fltype;  //filetype & attributes
        char flname[10]; //filename char
        flflag; //flag (unused)
        int flsectors; //no. of sectors the file is allocated
        unsigned char fltrack; //track where file starts
        char flsector; //sector where file starts.
        FAR flstart; //start address
        BIG fllength; //file length
        FAR flexecute; //line or address for execution
        char fldirnbr; //no. of subdir if it is a DIR item.
        char subdir; //code of subdir if file is in one
        char fllabel;
        char flother[4];
```

```
} FILE;
```

How to show a directory of the current disk.

```
#include "stdsv .h"

main() {
int i;
if (Dstat(disk 128)){
        i=0;
while (i!=Files){
Name (&FileTable[i++]);
putt (CR);
}
}
}
```

FLOAT * Fstat(FILENAME *f);
This function allows you to get info about the file in the struct f (file type & name), result is NULL if an error occurred, otherwise it is a pointer to struct FSTAT. How to improve it.

file = Fstat("\020auto* ") // to get info about BOOT file on disc...

If this returns NULL then the file does not exist (020 is octal representation of code for BASIC file type). Here is a description of FSTAT struct:

```
typedef struct {
        char (type;
        char fname[l0];
        FAR fstart;
        BIG (length;
        FAR fexecute;
        } FSTAT;
```

FAR* Path(FAR *a, int *len);
As per BASIC PATH$. Returns a pointer to a struct of type FAR, 'a' contains a full 19-bit address of path string and len variable contains the length of this string. MDOS only! usually path is "1:".

void SetDirectory(FAR * a, int len);
As BASIC's DIR="...." to set the current subdirectory. Parameter a, is a pointer to a FAR struct that contains a full l9bit address of path string. Parameter len is its length. Implicit is "1:"

FILE *Item(int n);
Returns a pointer to n file item in the buffer FileTable.

Calling : i=Item(9); is equivalent to i=&FileTable[9]; only it is faster.

void ErrorWindow(int err);
Displays an error window containing the error err (sometimes with an error description), waits for a key press, the screen is saved.

int OpenRead(FILE *f);
SV allows you to open only one file for writing and one for reading.
An example of this can be found on the disc, the file is called openfile.c. NOTE: the first parameter is a pointer to FILE, NOT FILENAME!! You must first read the directory using DstatQ, and use file item from FileTable[] buffer. You can read the file by using fgetc(stdfile) or other functions for file manipulation. OpenReadO does not use MDOS and allocates 16KB of memory fora buffer. The function returns:

0=ok-	ERR_OPEN_OK
1=no room for buffer-	ERR_OPEN_NOSPACE
2=a file is already open for reading-	ERR_OPEN_ALREADY
3=file not found	ERR OPEN NOTFOUND

int CloseRead(void);
A file that is open will be closed automatically with an EOF (End Of File) code, or by calling CloseRead.

int Open Write(FILENAME*f, int buffsize);
This function allows you to open a file for output. You can write to a file with the standard f. functions (fputc, fprintf, fputs...). The file must be closed after use. The function does not use DOS! The second parameter is ((MAXSIZE%16384)+1), the buffer will be created in memory, the whole file is in RAM e.g. If a file is upto 100KB, then ((100* 1024)%16384)+1=7. This parameter allows the best use of available memory. If a file is too long, it will be closed automatically. Results as previous function (OpenRead).

int CloseWrite(void);
Removes the buffer from memory and saves the file to disc.

void Name (FILENAME * file);
Prints the filename and type from the argument. e.g. "autoexec.b"

BYTE Err;
This is a variable that contains the error code of the last disc operation (NULL if OK),

Printer functions,

The interface allows output in two modes, text or graphics. The second mode is slower but uses standard BASIC font together with UDG and HUDG! The width, length and page margins are fully controlled. Graphics mode is standard EPSON ESC, "*",m,... mode, ESC is used as break.

void SetPrinter(int mode, int exp, int bold);

Sets the basic modes of the printer. If mode is 0, then text mode will be set, if mode is 1-8 then EPSON graphics mode 0 ... 7 will be set. The parameters exp (expanded) and bold are set to TRUE/FALSE (graphics only).

void InitPrinter(void);
Sends initialisation string and header to the printer.

void Outprinter(int c);
Sends char c directly to the printer.

void Lprint (int c);
Sends char c to the printer indirectly. The codes FF, CR and LF are controlled. Page design is controlled.

void Setpage(int pagelen, int linelen, int margin, int leftfeed);
This is used for fast sending o the page design:
pagelen = length of page
linelen = length of line
margin = left margin
leftfeed = code that will be sent after every CR

The Sound functions

SV recognises two common sound formats - E-Tracker and ProTracker modules. When the modules are in memory, the following functions allow you to play them.

void SetEtracker(FAR *a);
Plays a compiled E-tune. I recommend you compile the module from address 32768 49152. The parameter is a full 19bit address of the start of the module. The module is played automatically if interrupt is enabled (default).

void SetProtracker(FAR *a1, FAR * a2);
As the previous function.
ProTrackcr allows you to have the player and data in two different places, parameter al is the address of the player, a2 is the address of the data.

void MusicOn(void);
Void MusicOff(void);
Used to switch on/off sound modules.

void Beep (void);
void Click(void);

Some sound effects.

Functions to display numbers,

These functions are mainly for 19bit numbers in decimal or hexadecimal form.
The difference between designation of size (BIG type) and address (FAR type) is that RAM started at address 16384 and size from 0, bit 7 of offset is reset when size and set when address representation is used. e.g. representation of the number 32768,

address representation: page=l, offset=&8000
size representation: page=2, offset=&0000

If you want to show a number in the form of an address, you must increment the page by 1.

void Hex20(BIG *a);
void Printl9d(BIG *a);
The parameter is a pointer to a FAR or BIG struct. The functions show the number in hex or decimal form.

void Hex8(char c);
As the previous function, but in the range 0-255.

void Stream(int strm);
As per BASIC open x,"y" command. Allows you to open streams for in/out operations. SV uses Keyboard or file for input and printer or file for output.
The streams are marked as 2, 3 or 4 and are

stdout keyboard-> screen (default)
stdprn keyboard-> printer stdfile
disc file-> disc file

To work with streams use the f. functions (fprintf, fput etc..). For the parameters of Stream you use predefined constants stdout, stdprn & stdfile.

The keyboard and input functions,

This group of functions test for keyboard, ESC or joystick (mouse).

STRING Input (int line, int column, int max);
STRING Input2 (int line, int column, int max, STRING old);
Two very similar functions for string input. Line and column are the screen position for input and max is the maximum length of the string. Parameter old is a pointer to a string Louse as default. The result is NULL if escape is pressed, else a pointer to a buffer where the string is held. THE KEYS DEL, SYM DEL, CURSOR KEYS, ESC EDIT & RETURN are available.

void * Comput(int line, int column, int max, void *value, int sz);
Function to input numbers in decimal or hexadecimal base in range 0 to 999999 in 1, 2 or 3 byte form. The first three parameters are the same as Input(). value is a pointer to one of BIG, WORD or BYTE types, the last sz is size of it.

Returns NULL if ESC pressed.

BIG length;
char width;

Comput(0, 0, 7, &length, sizeof (length)); //l9bit
Comput(l, 0, 3, &length, sizeof (width)); //8bit

Use prefix "&" for hexadecimal numbers.

BOOL Esc(void);
Returns true is escape is pressed.

void Wait(UINT time);
Waits time/50 secs.

int Joystick(void);
Returns NULL if joystick #1 is not used, else bits are set

0- RIGHT
1- LEFT
2- DOWN
3- UP
4- FIRE (RETURN)
5-7 unused.

int getche (void);
Waits for a keypress and returns the ASCII form of it.

void Caps (BOOL fig);
Switch caps on/off.

void SetKeyboard(int repspd, int repdel);
Set keyboard autorepeat. Default is 16 and 5.

Functions for memory reserving

FAR *Reserved(BIG *a);
Function for memory allocation. The structure a contains the size of block that you want. Result is pointer to struct, it contains the address of the start of the block, NULL if no space is free.

void UnReserved(void);
Returns the last allocated block of memory to the system.

BIG *TestSpace(BIG *free);
Parameter is a pointer to an empty structure. Returns the size of all available memory as a full 19bit number. Returns NULL if there is no free memory.

Arithmetic functions

Most of these functions work with FAR (or BIG). These functions are for the manipulation of them.

FAR *Add20(FAR *result, FAR *op1 FAR *op2);
FAR *Sub20(FAR *result, FAR *op1, FAR *op2);
Add or subtract op2 with/ from op1. The result is stored in result, returns a pointer to it.

int Cp20(FAR *opt, FAR* op2); Compares 2 19bit numbers, returns : 0= numbers are equal1 if op I is greater than op2
-I if op2 is greater than opl.

BIG *Conv20(UINT value, BIG *value2);
Converts 16bit unsigned to 19bit number. The number is stored in value2 and returns a pointer to it.

BIG *Eval(BIG *nbr, STRING s);
As per Conv20(), except it evaluates string s as a number (0..999999; dec or hex).

Example:

```
BIG *start;
start= Eval(" ", "98304");
```

or: start=Conv20(50000, " ");

FAR ***Overpage(FAR *addr);**
This function is for normalisation of 19bit pointers after add or subtract.
E.g. address 03:D4F3 returns 04:94F3 etc

int Rand(void);
Returns pseudo-random number.

Functions for memory usage,

void Page (int page);
Switch page in sections C and D of RAM (address 32768 to 65535). You can only use this function from the low half of RAM (address 0 to 32767), else the program will crash.
This is important for every paging function!

void RePage(void);
Sets page in high memory (HMPR) to its original value.

void Scro(void);
void Scrf(void);
Switch in high memory VIDEORAM. Scrf() returns memory to its original state.

void Mode3(void);
void Mode4(void);
Set video mode to 3 or 4. Modes 1 and 2 are not available.

int SetTimer (DINT counter, void (*handler)());
SV allows you to use up to 16 objects of type TIMER, they are used to call procedures after a certain period of time. The time range is from 0 to 21 rains with steps of 0.02 sec. The first parameter is the time before execution, and is a multiple of 0.02, the second parameter is the address of your handler that will be called. The function will return the timer number that has been allocated (1...16), returns NULL if no timer is free. You can save this code for premature cancelation of the timer if you need to. After the function is called, the timer will be cancelled. The handler routine will be called with disable interrupt, without parameters. E.g SetTimer((50*60)*3,myfync); // calls myfunc() after 3 mins.

void KillTimer(int Tident);
Used if you want to cancel the timer before activation. The parameter is the timer ID code.

void ScreenSaver(int mins, void((*handle)());
You can set the time for a screen save routine (0-21 mins). If the second parameter is NULL then the system saver is used, otherwise your own routine is used. There is an example of this in the VIEW.C program.

void Linelnt(int scanline, void (*hanhle)());
Set line interrupt to screen microline and your own routine. Parameter scanline is from 0 to 191 (-1 for cancel). Handle is pointer to small and fast routine to change palette or mode.

The graphics functions,

void Dialogbox(int line, int row, int height, int width, int colour, int type, STRING text, STRING hots);
The first four parameters are for the position and size of the window. Colour is INK* 16 + paper, type is used to specify window design:
0- fat and simple frame
1- thin and simple frame
2- double frame
3- shaded frame.

Text is a string terminated by '\377' (chrS255), that will be printed as the windows label.
The next control codes you can use are
1-underline next char
16- change text colour, the next char should be INK* 16 +PAPER
22- change print position (AT), the next chars are line and column.
255- end of string.

The last parameter on the list is a list of keys that will be used as HOTHEYs for direct selection of menu item by keyboard. It is also terminated by '\377'. If you don't want any hotkeys then use parameter NULL.

After calling this function: the part of the screen that will be used is saved. When the window is displayed, a field is set up within the window where the mouse can be clicked. An example of this is the file PRNSETUP.C.

int Cursor(void)
You use this function together with DialogBox() and ClickField(). After calling the pointer (arrow) will be shown. You can move it along the screen by joystick, cursors+ return or mouse. If ESC is pressed then the function is terminated and the code 255 is returned. Right mouse button = ESC. If you click inside the actual window, or press one of the hotkeys the function returns a value from 0 to no of hotkey chars, or 15. Clicking out of the window is the same as pressing escape. If a hotkey is pressed, bit 7 is set in the return value. An example of this is PRNSETUP.C Cursor() also sets a global variable Temp1 which is the column relative to the start of the window where you have clicked.

EXAMPLE.

```
#include "stdsv  .h"

main() {
int option;
DialogBox (0,0,4,10,0xf0,2,"my window\377", "ABCQ\377");
printf ("\a\1\1\lA) option 1\a\2\1\1B) option\a\3\1\1C option\a\1\1Q quit");
while ((option = Cursor() & 0bl l 111 11)>3); //while not clicked inside
switch (option)
{
case 0:...

case3:
}
Closebox(); }
```

The example shows a window of size 4*10 chars, paper 0, ink 15, double frame with four items and underlined first char. Selecting the first item returns 0, second 1 etc.

void CloseBox(void);
void CloseAll(void);
These two functions are used to close the windows, the screen is restored. Unclosed windows will crash your program.
void ClickField (WINDOW *wnd);
If you don't want the default click field inside the whole window, you can change it. Parameter is a pointer to a structure that describes part of the screen in chars 16*32 in mode 4 or 16*64 in mode 3.
Description of WINDOW :

```
typedef struct{
char wtop; // top line (0 is top 15 is bottom)
```

```
char wleft; //start column
char wheight; // height of window (0...16)
char width ; // (0...32 or 0...64)
} WINDOW;
```

void Mouse (BOOL flg);
Switch on/off mouse controls, parameter is true/false type.

void ListBox (int line, int row, int height, int width, int colour, int type, STRING text,
void(*showitem)(int line, int row, int item),
int (*doitem)(int item),
int items);

If you want to see this function in action you must execute the example file VIEW. Listbox shows a window with various numbers of items(1 ... 255). The first seven parameters are equal to DialogBox(). Parameter items is maximum number of items, showitem is a pointer to a function that shows one item (item number is number of order) of menu at position line, row doitem is a pointer to a function that is called when an item is selected and parameter is the number of the item.

void Ink (int col);
void paper (int col);
Set current permanent colour.

void Cls(void);
Clear screen.

void Depack (FAR *addr);

void Gchar (int line, int row, int c);
Allows you to display some special chars at position. The chars are good for menu construction.

08 full circle
09 empty circle
10 square root sign
11 little cross
12^
13 arrow down
14 <-
15 ->

void at (int line, int row);
Set the print position (0..15 & 0..63).

void SetPalette (void);
Sends the current palette to the CLUT registers.

FAR * GetSprite(int line, int row, int hight, int width, FAR *a);
Similar to BASIC GRAB. It saves described part of the screen into memory at address in a. The result is a pointer to a. The global variable Nextspr contains the address after the last byte of he sprite.

void PutSprite(int line, int row, FAR *a);
Put grabbed sprite onto the screen.

NOTE GetSprite and PutSprite work in characters sizes and not pixels.

WINDOW *SetWindow(WINDOW *wnd);
Set a window on the screen for char output, as per BASIC

WINDOW *LookWindow(WINDOW *wnd);
Sets an empty struct wnd with current window co-ordinates. The result is again a pointer to wnd.

void SUp(void);
void SDown(void);
void SRight(void);
void SLeft(void);
Roll the current window about 1 char.

void SaveScrcen (int line, int row, int height, int width);
void LoadScreen(void);
Saves and loads a described part of the screen to a system screen buffer.

void FillBox (int line, int row, int height, int width, int byte);
void InvertBox (int line, int row, int height, int width);
Fill with byte or invert described part of the screen.

void Windowl(int line, int row, int height, int width, int colour);
void Windowl(int line, int row, int height, int width, int colour);
void Window2(int line, int row, int height, int width, int colour);
void Window3(int line, int row, int height, int width, int colour);
Simple procedures that show a window at position with size height, width and colour of type 0, 1, 2 or 3.

Memory moving

void Move (UINT from, UINT to, UINT size);
Moves the block of memory from address to a new address. Overlaying is prepared.

void Exchange(UINT from, UINT to, U1NT size);#
Exchanges two blocks of memory.

void FarMove (FAR *from, FAR *to, BIG size);
As par Move(), but works with all of RAM. Do not forget there is a difference between address 32768 (01:8000) and size 32768 (02:0000)!

void memset (UINT start, char value, UINT s);
Fill block of memory with char.

UINT Rom(int a, int bc, int de, int hl, int addr);
Calls a ROM routine with the values in registers. Returns the value from the hl register.

Unsigned Dpeek(FAR a);
void Dpoke(FAR a, unsigned value);
As per basic.

Global variables

FILE FileTable[80];
This array contains after first calling Dstat() info about every file on current disc.

char Err;
contains error code of last disc operations or NULL if OK.

char FillChar;
This variable can contain '', '0' a'\0'. This char will be used to fill blank chars for numbers shows. Normally = ' '.

char Hcad[30];
contains print page header of length 29 chars.

char PString[15];
String for printer initialisation. ESC,"@" normally.

char Margin;
left margin for printer

char LeftFeed;
char that will be send after CR

FAR Free;
Contain address of first *free* byte in SV system.

FAR Reserve;
Contain address of last free byte in SV system.

char CLUT[16]; current
system palette

char Disc;

current drive number (1, 2 or 3)

char Files;
Alter calling Dstat() contains number of files at disc
char PageLength;
char PageLength2;
char LineLength;
char LineLength2;

Permanent and temporary page design variable

int Temp1;
After calling Cursor() it contains column of click relative to top left corner of "Clickfield".

int Temp2;
After calling Cursor() it contains ASCII code of key if HOTKEY selection is used.

FAR nextspr;
After calling GetSprite() it contain address after sprite in memory.

STRING HotKey;
This pointer allows you to change the set of keys that are used like HOTKEY in DialogBox(). See DialogBox(), Cursor()…

```
// Standard SAM Vision header file
//
// Contents: function prototype
//           ERROR definitions
//           simple types definition
//           global variables declaration
//           etc.
//
// DATE of last modification : 31/08/1995
//
//
// ©1996 FRED Publishing
//
//           all rights reserved
//
//

#warning "SAM Vision 4.3 headers"

//
//
// The disc & files function
//
//
// name - is file name max. 10 chars long - char*
// for example : "D2:fred", "bill" etc.
//
// drive - is disc drive number (l, 2 or 3 for ramdisc) - int
//
// page - is page number (0..31) - int
//
// offset - is page offset (32768..49151 = &8000..&BFFF) -unsign
//
// len - is length of object - unsigned
//
// track - number of disc track (0..79; 128..207) - int
//
// sect - number of disc sector (1..10) - int
//
// addr - is standard 16-bit address - unsigned
//
// line - is print position line (0..15) - int
// column - is print position column (0..31 or 63 in mode 3)-int
// note : SV does not use standard 8*8 point char matrix, but
//           12*8 point (= not 24 ,but 16 line on screen)
//
// height - is height of object in characters (12) - int
```

```
//
// width - is width of objects in characters (8) - int
//
// ink - is colour of foreground (0..15) - int
//
// paper - is colour of background (0..15) - int
//
// colour - is one byte form of ink & paper = ink+(paper*256)-int
//
// text - is a string explicit terminated by chr$ (255) - char *
// note : Dialogbox & Listbox requested this, because PRINT AT
//              code (22) use params in range 0.... Strings are
//              terminated not with a \0 ,but \377 (octal 377 = 255 dec.)
//              example : char *str = "\a\000\000Top of screen\377";
 //
// hotkeys - is the definition of key for direct selection from menus
//              type of this is char *; terminated by \377 too
//              example : "ABCD\377"
//
// max - is maximal number limit - unsigned
//
// a, bc, dc, hl - is pseudo register variable - int
//
// value - is int
//
//---------------------------------------------------------------------

#define SV_VER 0x0403

typedef enum {
              BASIC = 16,
              ARRAYD,
              ARRAYS,
              CODE,
              SCREEN,
              DIR,
              HIDE =128,
              PROTECT=64 } TYPE;

typedef enum {
              BLACK,
              BLUE,
              RED,
              MAGENTA,
              GREEN,
              CYAN,
              YELLOW,
              WHITE,
```

```
        BBLACK,
        BBLUE,
        BRED,
        BMAGENTA,
        BGREEN,
        BCYAN,
        BYELLOW,
        BWHITE } COLOR;

typedef struct { char apage;
        unsigned aoffset;
        } FAR;

#define BIG FAR

typedef struct { char flype;
        char fname [10];
        FAR (start;
        BIG flength;
        FAR fexecute;
        { FSTAT;

typedef struct { char dnamc[10];
        char dtracks;
        int dident;
        BIG dused;
        } DSTAT;

typedef struct { char fltype;
        char flname[ l0];
        char flflag;
        unsigned int flsectors;
        unsigned char fltrack;
        unsigned char flsector;
        FAR flstart;
        BIG fllength;
        FAR flexecute;
        char fldirnbr;
        char flsubdir;
        char fllabel;
        char flother[4];
        } FILE;

typedef struct { unsigned char fntype;
        char fnname[10];
        } FILENAME;

typedef struct { char wtop;
        char wleft;
```

```
            char whcight;
            char wwidth;
            } WINDOW;

typedef unsigned UINT;
typedef unsigned char UCHAR;

typedef unsigned WORD;
 typedef unsigned char BYTE;

typedef int BOOL;

typedef char * STRING;

//-------------------------------------------------------------

// DISC errors const

#define ERRCODE        int

#define ERR_ESCAPE 84
#define ERR_BADTRACK 85
#define ERR_BADFORMAT 86
#define ERR_NODISC      87
#define ERR_BADTYPE    94
#define ERR_PROTECT    104
#define ERR_NOSPACE    105
#define ERR_DIRFULL    106
#define ERR_NOTFOUND 107

// ERROR code for R/W file fncs

#define ERR_OPEN_OK   0
#define ERR_OPEN_NOSPACE 1
#define ERR_OPEN_ALREADY 2
#define ERR_OPEN_NOTFOUND 3

extern struct DSTAT * Dstat(int drive);

extern struct FSTAT * Fstat(FILENAME *file);

extern ERRCODE Restore( void );


extern void Device( int drive);

extern FAR * Path(FAR *a, int *len);
extern void SetDirectory(FAR *a, int len);
```

```
extern ERRCODE Load(FILENAME *file);

extern ERRCODE Loadat(FILENAME *file, FAR *a);

extern ERRCODE Save(FILENAME *file, FAR *a, BIG *b);

extern ERRCODE Erase(FILENAME *file);

extern FILE * Item(int fileitem);

extern void ErrorWindow(ERRCODE error);

extern unsigned Rcad(int track, int sect, unsigned addr);

extern unsigned Write(int track, int sect, unsigned addr);

extern int OpenRead(FILE *file);

extern int CloseRead( void );

extern int OpenWrite(FILENAME *file, unsigned bufsize);

extern int CloseWrite( void );


// --------------------PRINTER FUNCTIONS--------------------

extern void InitPrinter( void );

extern void SetPrinter(int mode, int exp, int bold);

extern void Lprint(int c);

extern void OutPrinter(int c);

extern void SetPage(int pagelen, int linelen,
                    int margin, int leftfeed);



//-------------------SOUND FUNCTION-------------------------

extern void SetEtracker(FAR *a);

extern void SetPtracker(FAR *al, FAR *a2);

extern void MusicOff( void );
extern void MusicOn( void );
```

```
extern void Beep( void );

extern void Click( void);


// ---------- GRAPHICS & WINDOWS FUNCTION ----------


extern void DialogBox(int line , int column,
                      int height, int width,
                      int color , int type,
                      STRING txt, int hotkeys);

extern int ListBox(int line , int column,
                   int height, int width,
                   int color , int type,
                   STRING txt, int showitem,
                   int doitem, int items);

extern int ClickField(WINDOW *w); extern void

Mouse(BOOL flag); extern int Cursor( void );


extern void CloseBox( void );

extern void CloseAll( void );

extern void Window0(int line , int column,
                    int height, int width, int color);

extern void Windowl(int line , int column,
                    int height, int width, int color);

extern void Window2(int line , int column,
                    int height, int width, int color);

extern void Window3 (int line , int column,
                    int height, int width, int color);

extern WINDOW *LookWindow(WINDOW *w);

extern WINDOW *SetWindow(WINDOW *w);

extern FAR *GetSprite(int line , int column,
                      int height, int width, FAR *a);
extern void PutSprite(int line, int column, FAR *a);
```

```
extern void SaveScreen(int line,int column,int height,int widt);

extern void FillBox(int line , int column,
                int height, int width, int color);

extern void InvertBox(int line, int column,
                int height, int width);

extern void LoadScrcen( void );

extern void Depack(FAR *a);

extern void Cls( void );

extern void lnk(int ink);

extern void Paper(int paper);

extern void SUp( void );

extern void SDown( void );

extern void SLeft( void );

extern void SRight( void );


extern void Gchar(int line, int column, char c);

extern void at(int line, int column);

 extern void SetPalette( void );

//------------------PRINT STRING & NUMBER FUNCTIONS----------


extern void Hex8(char c);

extern void Hex20(BIG *a);

extern void Hex24(BIG *a);

extern void Printl9d(BIG *a);


extern void Name(FILENAME *file);

extern void Stream(int strm);

//-----------PAGING, INTERRUPTING, MODES ... ----------------
```

```
extern void Mode3( void );

extern void Mode4( void );

extern void Page(int page);

extern void Repage( void );

extern int SetTimer(int counter, void (*handler)());

extern void KillTimer(int nbr);

extern void ScreenSaver(int time, void (*handler)());

extern void Linelnt(int scanline, void (*handler)());

extern void Scro( void );

extern void Scrf( void );

extern WORD Rom(int a, int bc, int de, int hl, WORD addr);

extern FAR * Reserved(BIG *a);

extern void UnReserved( void );

extern BIG *TestSpace( BIG *a);

extern void Move(unsigned from, unsigned to, unsigned size);

extern void FarMove(FAR *from, FAR *to, BIG *size);

extern void Exchange(unsigned from, unsigned to, unsigned size);

extern void memset(unsigned from, unsigned with, unsigned size);

//---------------ARITHMETIC, ADDRESSING ... ----------------

extern int Random(int range);

extern FAR *Eva12O(FAR *a, STRING s);

extern FAR *Conv20(int value, FAR *a);

extern FAR *Add20(FAR *rslt,FAR *opl, FAR *op2);
extern FAR *Sub20(FAR *rslt,FAR *opl, FAR *op2);
```

```
extern int Cp20(FAR'opl, FAR *op2);

extern FAR*OvcrPage(FAR *a);

extern void Dpoke(FAR *a, WORD value);

extern WORD Dpeek(FAR *a);

 //-------------KEYBOARD, INPUTS ...-------------------

extern STRING Input(int line, int column, int max);

extern STRING Input2(int line, int column, int max, STRING s);

extern int Comput(int line,int colum,int max,void *value,int s);

extern BOOL Escape( void );

extern void Wait(int time);

extern int Joystick( void );

extern void SctKeyboard(int repdel, int repspd);

extern void Caps(BOOL fig);

//-----------------------------------------------------------------------

extern const BYTE LMPR, HMPR, VMPR;

extern FILE FileTable[80];

cxtcrn const BYTE Err;

extern BYTE FillChar;

extern BYTE PageWidth;

extern BYTE PageLength;

extern BYTE PageWidth2;

extern BYTE PageLength2;

extern BYTE Margin;

extern BYTE LeftFeed;
```

```
extern unsigned char Head[30];

extern unsigned char PString[15];

extern const FAR Free;

extern const FAR Reserve;

extern FAR nextspr;

extern unsigned char Clut[16];

extern BYTE Disc;

extern const char Files;

extern const int Temp1;

extern const int Temp2;

extern STRING HotKey;


#ifndef STDIO
#define STDIO      1
#define EOF        (-1)
#define ERR        (-2)
#define TRUE       1
#define FALSE      0
#define NULL       0
#define CR         13
#define LF         10
#define SPACE      ' '
#define stdout     2
#define stdprn     3
#define stdfile    4

/*
** These functions prototypes are built-in in the runtime file
*/

extern void putchar(int c);
extern void putc(int c);
extern void fputs(char *s, int strm);
extern void puts(char *s);
extern void fputc(int c, int strm);

extern int getchar( void );
extern int getc( void );
```

```
extern int getche( void );
extern int fgetc(int strrn);
extern char * fgets(char *string, int max, int stream);
extern char * gets(char *string);
extern int kbhit( void );

extern void Print(STRING s);

extern int printf(...); // these functions are VARIABLE!
extern int fprintf(...);
extern int sprintf(...);

extern void * MEMPTR_;          // pointer to first free byte
extern const unsigned stacklen_;
extern const unsigned heaplen_;

extern char * itoa(int value, char 'string, int radix);
extern char * itou(int value, char *string, int radix);
extern int atoi(char •s, int radix);
extern int isdigit(int c);
extern int isspace(int c);
extern unsigned strlen(char *s);

extern void assert(int e);
extern void abort( int error );
extern void exit( int val );
extern void * calloc( unsigned n, unsigned size);
extern void * malloc( unsigned size );
extern void * frcc(void *block);

#endif
```

Appendix **B**

On the SAM C disc are some examples of C programs for SAM Vision (as source and object modules). You should read the example source files.

TIME .C
Simple example to using of TIMERs. Shows time from 00:00:00. Routine works in real time on background !!!

DIR .C
This simple program shows the disc directory in ListBox window. You can use function Directory() and showfile() in your own progs.

PRNSET .C
This universal routine is for your programs that use the printer. Calling:
PmSetup(line, row, color, type);
Function shows menu at position window with color and types. It allows basic setting of SV print routine.

VIEW .C
This example is extensive and shows many of SV functions. You can preview files on disc 1..3. You can remove, get info and view files. Also uses a nice ScreenSaver. Program terminated by ESC pressing.

SPRITER .C
Program for sprite grabbing. Final product is in format for PutSprite().
1. load "spriter"
2. load your screen with sprite
3. goto 10 for run
4. cut sprite (marked top left corner and low right corner)
5. save sprite

OPENFILE.C
Uses input/output functions with disc files for practice.

SV_LOADER - LOADER FOR SV PROGS
SV40 .LIB - CODE OF LIBRARY
SV40 .S - SOURCE CODE FOR COMET OF IT
DEBUG2 - DEBUGGER FOR SV PROGRAMS
RUNTIME2.S - RUNTIME MODUE.
STDSV .Ii - GENERAL HEADERS FILE
CTYPE, STRING, STRING2, STDLIB, SYSTEM - OTHER VERSIONS OF STANDARD LIBRARIES

DIR .C - Source of examples
PRNSET .C

TIME .C
VIEW .C
OPENFILE.C
PC .C
SPRITER .C

SPRITER - Sprite Grabber
SCREENCOMP - Screen compression utility

SAM Vision pdf version 1
09-Jan-2005

PDF by Steve Parry-Thomas for Sam Users Everywhere.

www.samcoupe-pro-dos.co.uk

Published by
FRED Publishing, 40 Roundyhill, Monifieth, Dundee, DD5 4RZ